# Combiner Contraintes et Modèles pour le Traitement de Langage Contrôlé

**Mathias Kleiner** **Patrick Albert** **Jean Bézivin**

INRIA*, Centre Rennes - Bretagne Atlantique, Ecole des Mines de Nantes,
4 rue Alfred Kastler, 44307 Nantes, France
ILOG S.A, 9 rue de Verdun, 94253 Gentilly, France

mathias.kleiner@inria.fr

**Résumé**

Les schémas conceptuels (SC) sont des éléments centraux des systèmes d'information. Dans un processus d'administration, un problème récurrent et difficile est de permettre aux décideurs de directement définir et maintenir leurs schémas à l'aide d'un langage pseudo-naturel. Semantics for Business Vocabulary and Rules (SBVR), une spécification récemment publiée, offre une syntaxe abstraite permettant d'exprimer un SC et une syntaxe concrète à base d'Anglais structuré. Dans cet article, nous proposons une méthode originale pour extraire un modèle SBVR à partir d'un texte anglais puis le transformer en un diagramme de classe UML. Nous décrivons, dans un cadre d'ingénierie des modèles, une combinaison de programmation par contraintes orientée objet et de transformation de modèles. En plus des résultats théoriques, des expérimentations préliminaires sont fournies sur un exemple concret.

## 1 Introduction

Les schémas conceptuels (SC) sont largement utilisés dans l'industrie pour représenter de manière formelle les connaissances d'un système d'information. Un SC est souvent l'élément central sur lequel repose un ensemble d'opérations : validation, simulation, génération de cas d'utilisation, etc. La combinaison UML/OCL est le standard *de facto* pour la spécification de SC. Cependant la modélisation, la maintenance et l'évolution d'un SC requiert actuellement des compétences techniques importantes. Une approche récente en ingénierie logicielle cherche à faciliter ce processus en permettant aux décideurs d'exprimer leurs besoins en langage naturel pour ensuite les transformer dans une représentation formelle.

Dans le contexte commercial, l'Object Management Group (OMG) a récemment publié la spécification SBVR (Semantics for Business Vocabulary and Rules). SBVR offre un meta-modèle des concepts et des faits qui peut être utilisé pour définir un SC. La spécification propose également une syntaxe concrète sous la forme d'Anglais structuré. Cependant, l'analyse de langage naturel pour l'obtention d'un modèle SBVR est un problème difficile. En particulier, l'utilisation de grammaires *dépendantes du contexte* pour construire des langages contrôlés (structurés et spécifiques à un domaine) engendre la possibilité d'avoir des interprétations différentes d'une même phrase. Cette propriété disqualifie les algorithmes déterministes, comme les outils existant de l'ingénierie des modèles (IDM) (ATL[12], QVT[2]) et la plupart des *parseurs* de langages contrôlés (ACE[20]). Dans ces conditions, le problème devient combinatoire et requiert de *rechercher* des solutions à l'aide de techniques avancées comme celles utilisées en intelligence artificielle.

La principale contribution de cet article est une traduction automatique d'Anglais contrôlé en un modèle SBVR et un modèle UML du SC décrit, permettant de construire des langages flexibles (éventuellement spécifiques à un domaine). L'originalité de notre approche est qu'elle combine, dans un cadre d'IDM, les techniques de programmation par contraintes (PPC) et les outils de transformation de modèles. Elle est principalement composée de trois opérations. La première est une analyse syntaxique et grammaticale du texte, directement liée au domaine difficile du traitement du

*Team AtlanMod

langage : nous décrivons un parseur basé sur une approche de la PPC orientée objet appelée configuration [13]. La seconde tâche est une transformation du modèle résultant en un modèle SBVR. La troisième tâche est une transformation de ce modèle SBVR en un modèle UML du SC. Parallèlement, l'intégration de la PPC dans l'IDM en tant qu'outil avancé de transformation de modèles est une contribution importante et innovante de ce travail.

### 1.1 Plan de l'article

La Section 2 introduit brièvement les technologies employées dans notre approche. Nous présentons également une vue d'ensemble du processus et un exemple concret. La Section 3 introduit le parseur de langage. La Section 4 montre comment le modèle résultant est transformé en un modèle SBVR. La Section 5 propose une transformation de SBVR vers UML. L'implémentation et les expérimentations sont présentées en Section 6. Enfin, nous discutons de l'état de l'art en Section 7.

## 2 Contexte de travail

### 2.1 Brève introduction à SBVR

SBVR est un standard de l'OMG[3] dont le but est d'être une base pour la description d'activités commerciales en langage naturel. C'est une tentative pour combler l'écart entre les utilisateurs finaux et l'architecture logicielle, en permettant aux non-spécialistes de paramétrer et faire évoluer la logique commerciale de leurs applications.

SBVR standardise un ensemble de concepts permettant de définir des langages contrôlés (langages déclaratifs dont la grammaire et le lexique sont limités pour éliminer une partie de l'ambiguïté). Voir [20] pour un article historique, et plus récemment [15]). Les systèmes de règles métier comme ILOG JRules ou Drools sont des langages contrôlés populaires utilisés pour modéliser explicitement la logique commerciale dans un nombre croissant d'applications.

Nouns ne décrirons pas SBVR de manière exhaustive dans cet article. Cependant les figures 1 et 2 illustrent la sophistication des meta-modèles et l'approche qui consiste à séparer les formulations logiques des concepts et des faits.

### 2.2 Brève introduction à l'IDM et la transformation de modèles

L'ingénierie des modèles est un domaine de recherche émergeant qui considère les artefacts logiciels principaux comme des graphes typés. Cela provient du besoin industriel d'avoir une organisation régulière et homogène où les différents aspects d'une architecture logicielle peuvent être séparés ou combinés.

En IDM, les modèles sont le concept unificateur. La communauté utilise les concepts de modèle terminal, méta-modèle et méta-méta-modèle. Un modèle terminal est la représentation d'un système. Il capture certaines de ses caractéristiques et fournit des connaissances. Les outils de l'IDM agissent sur des modèles terminaux exprimés dans des langages précis de modélisation. La syntaxe abstraite d'un langage de modélisation est appelée méta-modèle quand elle est exprimée elle-même comme un modèle. La relation entre un modèle et le méta-modèle de son langage est appelée *conformsTo*. Les méta-modèles sont eux-mêmes exprimés dans un langage de modélisation dont les bases conceptuelles sont capturées par un modèle auto-descriptif appelé méta-méta-modèle. L'architecture résultante est composée de trois niveaux appelés M1, M2, M3. Une définition formelle de ces concepts peut-être trouvée dans [11]. Les principes de l'IDM peuvent être implémentés dans différents standards. Par exemple, l'OMG propose un méta-méta-modèle standard appelé Meta Object Facility (MOF).

L'automatisation de l'IDM est principalement assurée par des méthodes de transformation. La production d'un modèle Mb à partir d'un modèle Ma par une transformation Mt est appelée transformation de modèle. Quand le méta-modèle source est identique à la cible (MMa = MMb), la transformation est dite endogène. Sinon, la transformation est exogène. Dans ce travail nous utilisons ATL (AtlanMod Transformation Language), un langage [12] de type QVT permettant une expression déclarative de la transformation à travers un ensemble de règles.

### 2.3 Brève introduction à la configuration

Configurer consiste à composer un système complexe à partir de composants génériques[13]. Les composants, aussi appélés objets ou éléments de modèle dans la suite, sont définis par leur type, attributs et relations mutuelles. Les systèmes acceptables, spécifiés par des contraintes, sont de plus restreints par la requête : un ensemble de besoins spécifiques, représentés par un fragment du système souhaité (i.e des objets interconnectés). Du point de vue de la représentation des connaissances, la configuration peut-être assimilée à la recherche d'un graphe typé et attribué (la structure) obéissant aux restrictions d'un modèle objet sous contraintes. Du point de vue de l'IDM, le problème est de trouver un modèle fini conforme à un méta-modèle. Plus précisément, nous considérons le processus comme une transformation de modèle dans

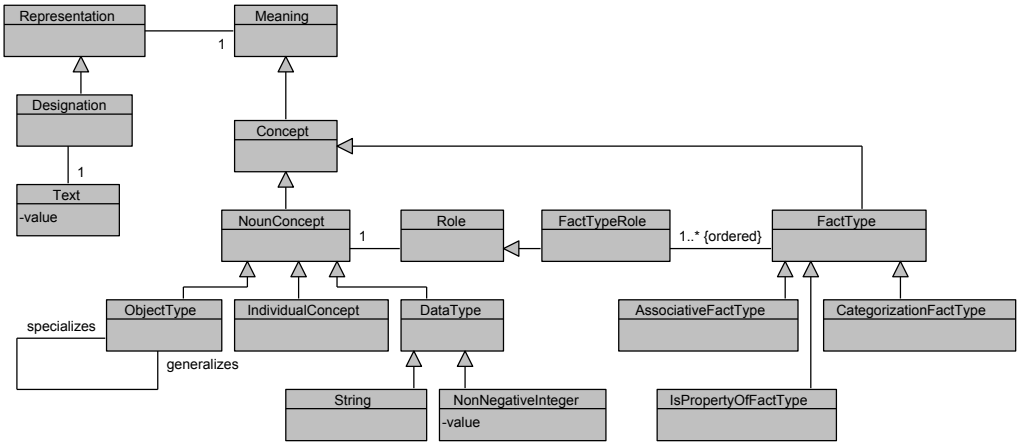


FIGURE 1 – Extrait du meta-modèle SBVR : concepts et faits

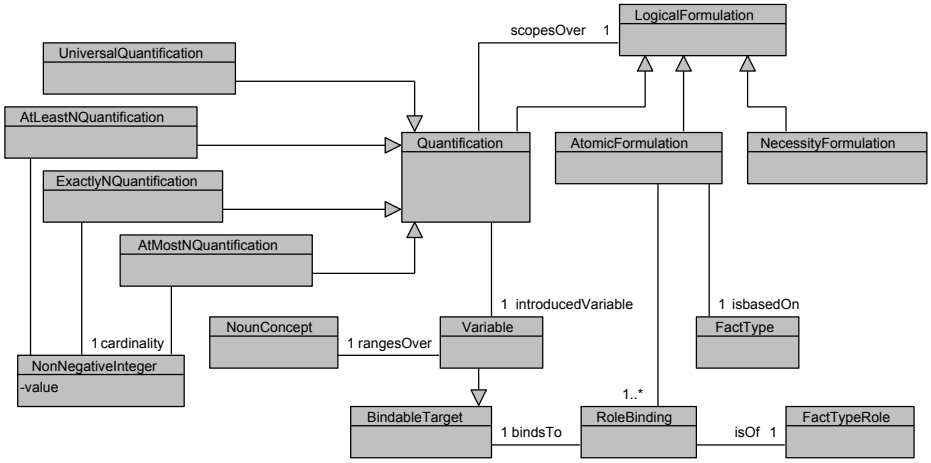


FIGURE 2 – Extrait du meta-modèle SBVR : formulations logiques

laquelle le modèle source est la requête et le modèle cible la solution. Le modèle de configuration joue alors le rôle de méta-modèle cible. Une version *relaxée* de ce méta-modèle définit le méta-modèle source. Elle est obtenue par le retrait de toutes les contraintes. Dans ces conditions, la requête (un modèle cible incomplet) est conforme au méta-modèle source et la configuration peut-être insérée comme une technique avancée de transformation de modèle : le configurateur *recherche* un modèle, en complétant la source et en y ajoutant tous les éléments de modèle nécéssaires pour que le résultat soit conforme aux contraintes du méta-modèle cible.

Plusieurs techniques ont été proposées pour traiter des problèmes de configuration : extensions des CSP (Constraint Satisfaction Problem) [18, 23, 19], approches à base de connaissances [22], programmation logique [21], approches orientées objet [17, 14]. La configuration est traditionnellement utilisée dans des applications industrielles comme la simulation de production. Plus récemment, les capacités expressives du formalisme ont montré son utilité pour la résolution de problèmes d'intelligence artificielle comme le traitement du langage [8]. Une introduction plus poussée à la configuration peut être trouvée dans [13].

Dans la suite nous proposons d'utiliser Ilog JConfigurator[14] pour l'analyse de langage contrôlé vers SBVR. Dans cette approche, un modèle de configuration (dans notre contexte, le méta-modèle cible) est bien défini par un ensemble de classes, relations et contraintes. Le langage UML/OCL peut être utilisé pour ces spécifications [7].

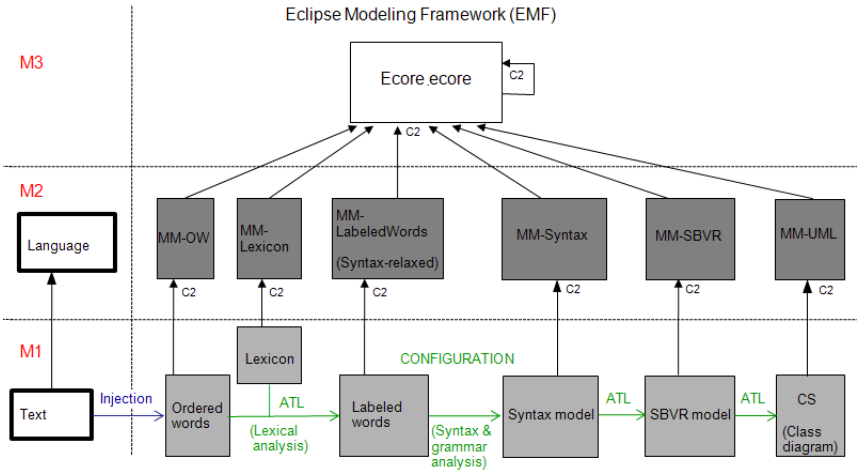


FIGURE 3 – Vue d'ensemble du processus

### 2.4 Vue d'ensemble du processus

La figure 3 montre l'ensemble du processus dans un cadre d'IDM. L'entrée est un texte Anglais, proche de la forme structurée proposée par le standard SBVR [3]. Le texte est injecté en modèle dont le méta-modèle est une simple annotation de la position des mots et des phrases dans le texte. Une transformation triviale utilise un lexique pour labeliser chaque mot par un ensemble de ses catégories syntaxiques possibles. Nous définissons ensuite un méta-modèle, appelé *Syntax*, pour lequel nous avons adapté les grammaires de configuration [8] à SBVR et le contexte de l'IDM. Le texte (en tant que mots labelisés) est fourni au configurateur grâce à la version relâchée de Syntax. Le résultat de cette analyse syntaxique et grammaticale est donc un modèle fini conforme au méta-modèle Syntax. Ce modèle est ensuite transformé avec ATL en un modèle conforme au méta-modèle SBVR grâce à un ensemble de règles utilisant les dépendances grammaticales générées. Ce modèle terminal SBVR peut ensuite être de nouveau transformé via ATL pour obtenir un modèle UML correspondant.

### 2.5 Exemple

Un exemple sera utilisé tout au long de cet article pour illustrer l'approche. Le texte considéré est composé de trois phrases définissant un SC :
*(1) Each company sells at least one product.*
*(2) Each product is sold by exactly one company.*
*(3) A software is a product.*
Cet exemple relativement simple contient pourtant des concepts non triviaux. Du point de vue du langage, nous utilisons des noms, des verbes à la forme passive ou active, ainsi que différents quantificateurs. Les phrases sont liées par le contexte car elles décrivent différents aspects des mêmes concepts. Du point de vue de la modélisation l'exemple décrit les notions de classes, d'héritage et de relations contraintes. Dans les sections qui suivent, nous montrerons l'application de chaque opération sur ces phrases.

## 3 Traitement de langue Anglaise contrôlée pour SBVR

Le traitement du langage naturel est un défi majeur de l'intelligence artificielle. Au vu de la difficulté du problème, de nombreux efforts ont été portés vers le domaine plus accessible des langage contrôlés [16], où des ambiguïtés sont retirées et le vocabulaire restreint. Parmi les approches existantes, [8] a montré que les grammaires de propriétés [4] peuvent être capturées dans un modèle de configuration afin de traiter un sous-ensemble de la langue française. Le parseur résultant n'hérite pas du comportement déterministe de la plupart des approches et est prévu pour être adapté à différentes grammaires. Nous avons modifié et étendu cette méthode pour l'Anglais et SBVR. Dans notre cadre d'IDM, le modèle de configuration est défini en tant que méta-modèle (Syntax).

### 3.1 Méta-modèle Syntax

Un fragment de ce méta-modèle (principalement les classes et les relations) est présenté dans la figure 4. Syntax capture trois types d'information à partir du texte : les catégories syntaxiques, les dépendances grammaticales et la sémantique SBVR.

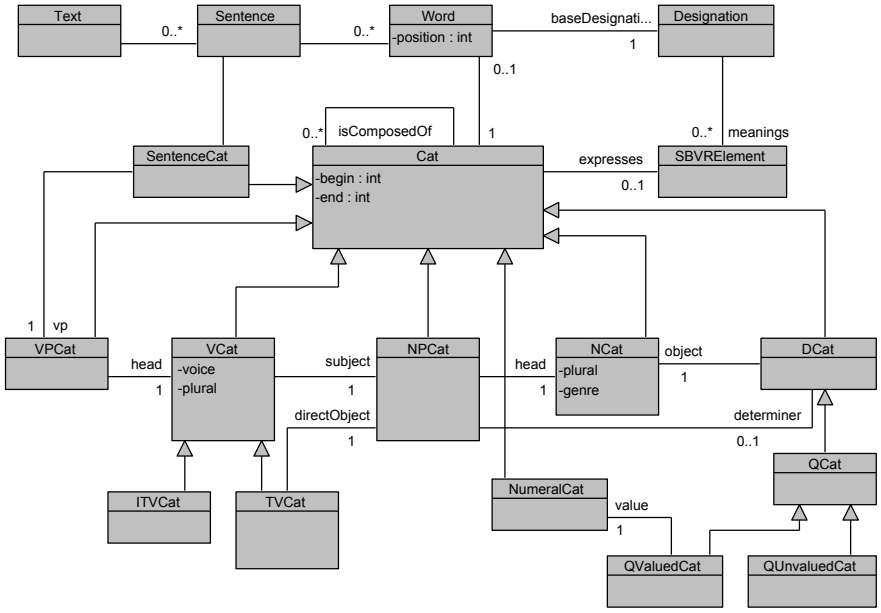


FIGURE 4 – Extrait du méta-modèle Syntax : syntaxe et grammaire

### 3.1.1 Analyse syntaxique

Afin d'obtenir un arbre syntaxique à partir d'une phrase, nous avons adapté le modèle des grammaires de propriété à l'Anglais. La classes principale est *Cat* et représente une catégorie syntaxique. Une catégorie est *terminale* quand elle est directement associée à un mot. Ces catégories incluent *NCat* (nom), *VCat* (verbe) ou *DCat* (déterminant). Ces catégories peuvent être spécialisées : un verbe est soit transitif (*TVCat*) ou intransitif (*ITVCat*). Les catégories possibles d'un mot sont obtenues grâce au lexique par la transformation précédente. Une catégorie est *non-terminale* quand elle est composée d'autres catégories. *SentenceCat* (phrase), *NPCat* (groupe nominal), *VPCat* (groupe verbal) sont les catégories non-terminales principales. Un ensemble de contraintes définit les catégorisations acceptables. Ces contraintes impliquent par exemple les constituants ou leur position relative dans une phrase. Voici quelques exemples de ces contraintes en OCL :

- *Chaque groupe verbal dont le verbe est transitif est composé d'au moins un groupe nominal.* Cette contrainte s'applique à la relation *isComposedOf* d'une catégorie :

```
context VPCat
inv: head.oclIsTypeOf(TVCat)
        implies isComposedOf->
          exists( elt.oclIsTypeOf(NPCat) )
```

- *Un groupe verbal est toujours précédé d'un groupe nominal.* Cette contrainte s'applique aux attributs *begin* et *end* des catégories (obtenus par la position des mots associés) :

```
context SentenceCat
inv: isComposedOf->exists(
        elt.oclIsTypeOf(NPCat)
        and elt.end < vp.begin )
```

### 3.1.2 Dépendances grammaticales

Nous avons étendu le modèle syntaxique pour faire apparaître explicitement les dépendances grammaticales en tant que relations entre les catégories. De la même manière, un ensemble de contraintes définit les constructions possibles. Voici quelques exemples de ces contraintes spécifiées en OCL :

- *Le sujet d'un verbe actif est placé avant le groupe verbal* :

```
context VPCat
inv: (head.voice = 'active')
        implies head.subject.end < begin
```

- *Un verbe et la tête de son sujet partagent le même pluriel* :

```
context VCat
inv: plural = subject.head.plural
```

### 3.1.3 Sémantique SBVR

Le meta-modèle est également enrichi par les principaux concepts de SBVR. Cette sémantique est assignée aux catégories grâce à la relation *expresses*. De nouveau, les contraintes gouvernent les affectations possibles. Quelques exemples :

– *Un verbe transitif exprime un* FactType :

```
context TVCat
inv: not expresses.oclIsUndefined()
        and expresses.oclIsKindOf(FactType)
```

– *La tête du sujet d'un verbe exprime soit un* ObjectType *soit un* IndividualConcept :

```
context VCat
inv: subject.head.expresses.
       oclIsKindOf(ObjectType)
     or subject.head.expresses.
       oclIsKindOf(IndividualConcept)
```

**A propos de l'unicité des concepts SBVR** Une difficulté majeure dans l'assignation de sémantique SBVR réside dans l'unicité des concepts. Plus précisément, le même concept SBVR peut être exprimé dans plusieurs phrases (voire la même). Considérons les deux premières phrases de notre exemple : les concepts "Company", "Product" et "To sell" sont exprimés plusieurs fois. Il faut éviter de dupliquer les éléments SBVR dans le modèle résultat. Un ensemble de contraintes force l'unicité de ces éléments sur la base d'hypothèses d'équivalence. Dans le cas d'éléments de la classe *NounConcept*, la désambiguation est faite sur la désignation des mots. Elle peut être formalisée en OCL de la manière suivante :

```
inv: NCat.allInstances()->
       forAll(n1,n2 : NCat |
       (n1.word.baseDesignation =
        n2.word.baseDesignation)
       = (n1.expresses = n2.expresses))
```

Puisque la désignation *canonique* est utilisée, différentes formes du même mot sont reconnues (i.e "products" et "product"). Le même principe est appliqué à d'autres éléments SBVR tels que les *FactType*.

### 3.2 Processus d'analyse et résultat

Comme expliqué précédemment, l'entrée du processus de configuration est un modèle d'une version relâchée du méta-modèle cible. Dans notre contexte, l'entrée est un ensemble d'objets inter-connectés de type *Text*, *Sentence*, *Word*, *Designation* et *Cat*. En effet, la transformation précédente, basée sur un lexique, a fourni les propriétés de chaque mot (pluriel, genre), leur désignation canonique ("has" a pour base "to have"), et les catégories syntaxiques candidates (le mot "one" peut être un nom, un nombre ou un adjectif).

Le résultat du processus de configuration est un (ou plusieurs) modèle(s) terminal(aux) satisfiant les contraintes, quand un tel modèle existe. La Figure 5 montre (un fragment) du modèle généré pour la phrase "Each company sells at least one product".
Il faut noter que ce processus n'est pas déterministe : à cause des ambiguïtés du langage, plusieurs solutions peuvent êtres valides pour une même requête. Par exemple, considérons la phrase "MyCode is a software". Sans information lexicale ou contexte, il n'est pas possible de décider si le *NounConcept* "MyCode" est un *ObjectType* (une spécialisation de "Software") ou un *IndividualConcept* (une instanciation de "Software"). Plutôt que de décider arbitrairement de restrictions dans le langage, notre méthode permet de générer toutes les solutions valides afin de les comparer, ou même d'optimiser le modèle cible sur la base de préférences. De ce point de vue, notre approche offre une flexibilité supérieure à la plupart des analyseurs.

## 4 Transformation du modèle syntaxique en un modèle SBVR

Le modèle produit par l'analyse exhibe la sémantique SBVR exprimée par des (groupes de) mots. Grâce à ces informations et les dépendances grammaticales entre les catégories, il est possible de construire un modèle SBVR complet du texte fourni. Cette opération est réalisée avec ATL[12], un outil de transformation de modèles dans lequel la mise en correspondance est exprimée par des règles déclaratives entre un méta-modèle source (ici, Syntax) et un méta-modèle cible (ici, SBVR). Nous ne présentons pas exhaustivement chaque règle de la transformation mais nous donnons ci-dessous ses principaux ressorts. [1]

### 4.1 Vue d'ensemble de la transformation

Une première mise en correspondance est évidente : chaque *SBVRElement* de Syntax a son équivalent dans SBVR et implique donc la création de cet élément cible. Cependant, les relations entre les éléments SBVR ne sont pas explicites dans le modèle source et peuvent nécessiter des éléments intermédiaires. Un ensemble de règles permet de les dériver à partir des relations grammaticales.

Par exemple, considérons la règle suivante qui génère un *AssociativeFactType* (binaire) et ses rôles à partir d'un verbe transitif, son sujet et son complément direct. Elle peut s'écrire de manière informelle : "Pour tout AssociativeFactType B exprimé par un verbe V du modèle source, créer un AssociativeFactType B', avec deux roles R1 et R2, où le concept de R1 est le concept cible du sujet de V et le concept de R2 est le concept cible du complément de V". La règle crée des éléments intermédiaires (les rôles) et les utilise pour mettre en relation les concepts SBVR. Certains de ces

1. Le code source et la documentation des transformations présentées dans cet article ont été soumis en tant que contribution Eclipse au projet ATL et sont disponibles sur http ://www.eclipse.org/m2m/atl/

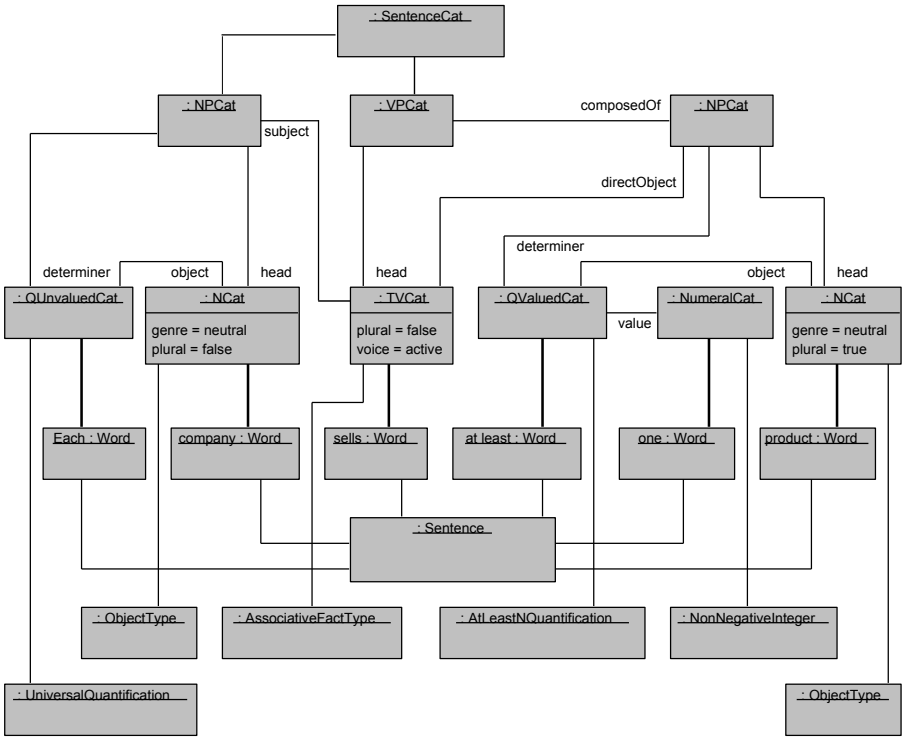


FIGURE 5 – Exemple : extrait d'un modèle terminal de Syntax

éléments (concepts cibles du sujet et du complément) étant eux-même crées par une autre règle.

La transformation permet également de créer des valeurs d'attributs à partir d'une source d'information d'un type différent. En effet, dans la première phrase de notre exemple, le mot "one" est associé à la catégorie *NumeralCat* et exprime un entier (*NonNegativeInteger*). La règle qui crée l'élément cible affectera la valeur 1 à l'attribut *value* de type *Integer*.

### 4.2 Processus de transformation et résultat

Une fois la transformation réalisée nous obtenons un modèle conforme à SBVR qui ne possède plus d'information syntaxique. La Figure 6 montre (un fragment) du modèle SBVR généré pour la première phrase de notre exemple.

## 5 Transformation du modèle SBVR en un modèle UML

Le modèle SBVR généré peut ensuite être transformé avec ATL en un modèle UML[**?**] correspondant au SC. De nouveau, nous ne présentons pas chaque règle mais les principales mises en correspondance.

### 5.1 Vue d'ensemble de la transformation

Quelques exemples sont présentés dans la Table 1 où la notation pointée est utilisée pour naviguer à travers les attributs et les relations. Ces correspondances sont assez naturelles : un type d'objet devient une classe, un type de fait associatif devient une association, une catégorisation désigne l'héritage (Generalization en UML), une propriété de type de fait se réfère à un attribut, un concept individuel devient une instance, etc. Les liens entre les concepts et les valeurs sont également explicites. Cependant, la plupart des règles ne réalisent pas une transformation unaire triviale. Par exemple, le concept SBVR *AtLeastNQuantification*. La propriété (*Property*) pour laquelle la valeur minimale (*lowerValue*) est affectée est celle obtenue par la transformation d'un fait (*AssociativeFactType*) cible de la relation *AtLeastNQuantification.scopesOver.isBasedOn*.

### 5.2 Processus de transformation et résultat

Une fois la transformation réalisée, nous obtenons une spécification UML du SC. La Figure 7 montre un modèle UML terminal pour notre exemple.

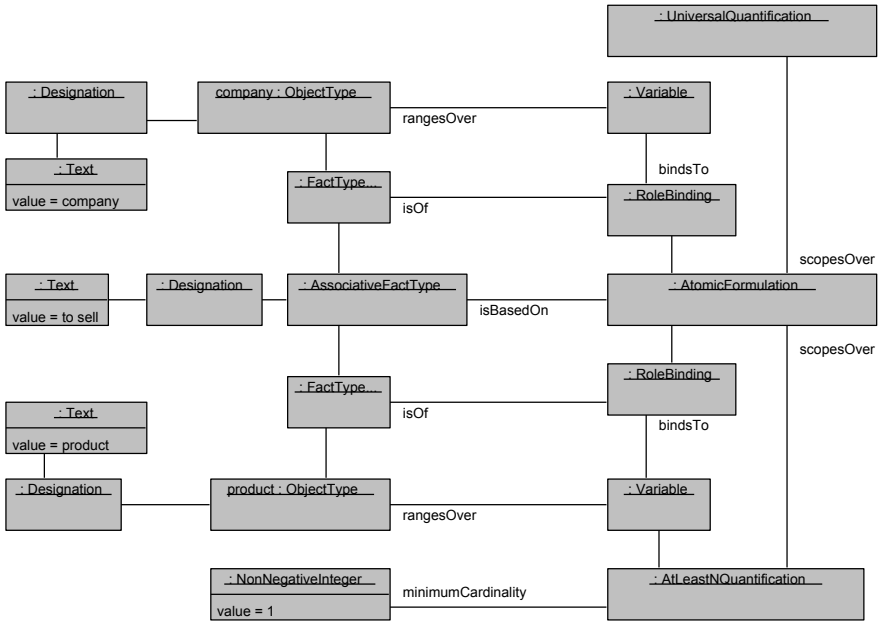


FIGURE 6 – Exemple : extrait d'un modèle terminal de SBVR

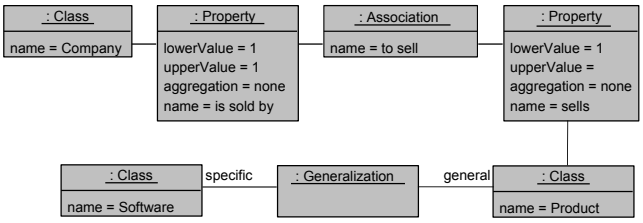


FIGURE 7 – Exemple : extrait du modèle UML

| phrase(s) | *Temps* | *Vars* | *Contraintes* |
|---|---|---|---|
| (1) | 0.26 | 527 | 1025 |
| (2) | 0.20 | 526 | 1022 |
| (3) | 0.19 | 475 | 885 |
| (1)+(2) | 1.82 | 973 | 2819 |
| (1)+(2)+(3) | 5.21 | 1328 | 5312 |

TABLE 2 – Expérimentations sur l'exemple (temps en secondes)

# 6 Implémentation et expérimentations

## 6.1 Implémentation

L'approche a été intégrée dans un cadre d'IDM basé sur Eclipse. Pour cela, chaque meta-modèle présenté a été défini grâce au langage de méta-modélisation KM3 [11], qui offre une conversion automatique vers le format ECore de l'EMF [1]. Ces méta-modèles ECore sont les sources et les cibles des transformations ATL proposées. Le configurateur (JConfigurator) a son propre langage. Le meta-modèle Syntax est donc défini également dans l'outil en tant que modèle de configuration. A l'execution, le modèle représentant la requête est extrait vers XML pour l'analyse, et la solution XML est injectée en un modèle de Syntax au format ECore. Ce modèle, au format XMI, est ensuite transmis aux transformations ATL.

## 6.2 Expérimentations

Nous présentons des expérimentations préliminaires sur l'exemple proposé, conduites sur un Intel Core2Duo 3Ghz - 3GB RAM. La Table 6.2 montre les résultats. Nous avons d'abord analysé chaque phrase séparément puis plusieurs phrases à la fois. Seuls les temps du configurateur sont affichés car les transformations ATL se déroulent en un temps négligeable (moins de 0.2 secondes).

L'analyse est efficace pour chaque phrase individuellement, mais le temps requis pour la recherche croit rapidement avec le texte complet. Ceci est dû à la taille du modèle source qui impacte directement l'espace de recherche du configurateur. D'un autre côté, les transformations ATL peuvent gérer de larges modèles. La séparation des tâches est donc positive pour la performance et pourrait être développée afin de réduire la

| SBVR concept | UML Concept |
|---|---|
| ObjectType | Class |
| ObjectType.Designation.Text.value | Class.name |
| DataType | DataType |
| IndividualConcept | InstanceSpecification |
| AssociativeFactType | Association |
| AssociativeFactType.Designation.Text.value | Association.name |
| AssociativeFactType.FactTypeRole#1 | Property (Association.memberEnd) |
| IsPropertyOfFactType.FactTypeRole#1.nounConcept | Property.classifier |
| CategorizationFactType | Generalization |
| CategorizationFactType.FactTypeRole#1 | Generalization.general |
| AtLeastNQuantification.minimumCardinality.value | Property.lowerValue |

TABLE 1 – Extrait du mapping entre les concepts SBVR et UML

complexité du modèle de configuration au minimum requis pour l'analyse syntaxique. Une autre alternative envisagée est d'analyser le texte de manière incrémentale, phrase par phrase, puis d'utiliser les capacités multi-source d'ATL pour unifier les modèles SBVR résultats. Il faut cependant noter que ce sont des expériementations préliminaires sur un problème connu pour être difficile. Aucune optimisation (heuristiques, propagation ou élimination de symétries) n'a été appliquée au moteur de configuration. Ces techniques sont fréquemment employées pour réduire les temps de calcul des problèmes sous contraintes. L'analyse réussie de notre exemple montre cependant la faisabilité de l'approche.

## 7 Travaux relatifs

[10] décrit la transformation inverse : à partir d'une description UML/OCL d'un SC, les auteurs montrent sa transformation en un modèle SBVR (via ATL), puis son paraphrasage en un texte Anglais structuré. La combinaison des deux approches est prometteuse. En effet, la conception d'un SC requiert généralement de nombreuses discussions entre décideurs et concepteurs, et la maintenance d'un SC amène de nombreuses évolutions. La combinaison pourrait permettre de traduire automatiquement les modifications d'une réprésentation vers une autre.
Des recherches précédentes ont été conduites sur l'utilisation des techniques de programmation par contraintes pour l'IDM, principalement sur l'animation de spécifications relationnelles ou la vérification de modèles. [5] transforme des spécifications UML/OCL en un problème CSP afin de vérifier la satisfaisabilité. [6] propose une méthode similaire, bien que le solveur (Alloy [9]) utilisé soit basé sur SAT. Les deux approches héritent des limitations du formalisme vers lequel les spécifications sont traduites, alors que le paradigme de la configuration est suffisamment expressif pour capturer directement la représentation sous forme de modèles. A notre connaissance, aucun travail existant n'a proposé l'utilisation de la recherche à base de contraintes comme outil de transformation pour l'IDM. En ce qui concerne le domaine de l'analyse de langages naturels contrôlés, notre approche diffère de la plupart des méthodes existantes (comme ACE[20]). En effet, ces parseurs n'acceptent pas de grammaire aimbigües, alors que nous pouvons paramétrer le niveau d'ambiguïtés acceptées. Ceci nous permet de mettre en balance la couverture du langage et l'efficacité computationnelle.

## 8 Conclusion et perspectives

Nous avons décrit une méthode qui permet d'analyser la spécification d'un SC, exprimée sous la forme de texte Anglais, et de la transformer un modèle de classe UML. Le standard de l'OMG SBVR est utilisé comme niveau intermédiaire de réprésentation. L'originalité de notre approche réside dans l'utilisation d'une technique avancée de programmation par contraintes, la configuration orientée-objet, en tant qu'outil de transformation de modèles intégré dans une architecture d'IDM. Des expérimentations préliminaires sont fournies pour vérifier la faisabilité de l'approche. De plus, l'analyseur présenté est flexible vis à vis de la couverture du langage et de l'ambiguïté acceptée, permettant ainsi de construire des langages contrôlés (spécifiques à un domaine) qui ne soient pas restreints à des grammaires libre de contexte. Ce travail offre plusieurs perspectives. Tout d'abord, les méta-modèles peuvent être étendus pour capturer un fragment plus important de la langue et de la sémantique SBVR. La génération de contraintes OCL, en plus d'UML, sera probablement nécessaire pour exprimer des sémantiques plus complexe. D'autres formalismes, tels qu'OWL ou

les systèmes à base de règles, peuvent être envisagés comme cibles finales du processus. Enfin, l'utilisation de la configuration peut être utile pour d'autres opérations complexes de l'IDM, dans lesquelles les transformations à base de règles déterministes ne sont pas suffisantes, et qui nécessitent de rechercher un modèle satisfaisant ou optimal parmi un ensemble de solutions potentielles.

## Références


[1] Emf ecore : http ://www.eclipse.org/modeling/emf.

[2] Qvt specification : http ://www.omg.org/docs/formal/08-04-03.pdf.

[3] Semantics of business vocabulary and business rules (sbvr) 1.0 specification : http ://www.omg.org/spec/sbvr/1.0/.

[4] Philippe Blache and Jean-Marie Balfourier. Property grammars : a flexible constraint-based approach to parsing. In *IWPT*. Tsinghua University Press, 2001.

[5] Jordi Cabot, Robert Clarisó, and Daniel Riera. Umltocsp : a tool for the formal verification of uml/ocl models using constraint programming. In R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer, editors, *ASE*, pages 547–548. ACM, 2007.

[6] Trung T. Dinh-Trong, Sudipto Ghosh, and Robert B. France. A systematic approach to generate inputs to test uml design models. In *ISSRE*, pages 95–104. IEEE Computer Society, 2006.

[7] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Configuration knowledge representation using uml/ocl. In Jean-Marc Jézéquel, Heinrich Hußmann, and Stephen Cook, editors, *UML*, volume 2460 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2002.

[8] Laurent Henocque and Mathieu Estratat. Parsing languages with a configurator. In *Proceedings of the European Conference for Artificial Intelligence ECAI'2004*, pages 591–595, Valencia, Spain, August 2004.

[9] Daniel Jackson. Automating first-order relational logic. In *SIGSOFT FSE*, pages 130–139, 2000.

[10] Ruth Raventós Jordi Cabot, Raquel Pau. From uml/ocl to sbvr specifications : a challenging transformation. *Information Systems Elsevier Journal*, 2009.

[11] Frédéric Jouault and Jean Bézivin. Km3 : A dsl for metamodel specification. In Roberto Gorrieri and Heike Wehrheim, editors, *FMOODS*, volume 4037 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2006.

[12] Frédéric Jouault and Ivan Kurtev. Transforming models with atl. In Jean-Michel Bruel, editor, *MoDELS Satellite Events*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138. Springer, 2005.

[13] Ulrich Junker. *Configuration*, volume Handbook of Constraint Programming, chapter 26. 2006.

[14] Ulrich Junker and Daniel Mailharro. The logic of (j)configurator : Combining constraint programming with a description logic. In *proceedings of IJCAI'03*, Acapulco, Mexico, 2003. Springer.

[15] Kaarel Kaljurand. Ace view - an ontology and rule editor based on controlled english. In Christian Bizer and Anupam Joshi, editors, *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[16] R. I. Kittredge. *Sublanguages and controlled languages*. Oxford University Press, 2003.

[17] Daniel Mailharro. A classification and constraint-based framework for configuration. *AI in Engineering, Design and Manufacturing, (12)*, pages 383–397, 1998.

[18] Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, pages 25–32, Boston, MA, 1990.

[19] Daniel Sabin and Eugene C. Freuder. Composite constraint satisfaction. In *Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[20] Rolf Schwitter and Norbert E. Fuchs. Attempto controlled english (ace) a seemingly informal bridgehead in formal territory (poster abstract). In *JICSLP*, page 536, 1996.

[21] Timo Soininen, Ilkka Niemela, Juha Tiihonen, and Reijo Sulonen. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring Symp. on Answer Set Programming : Towards Efficient and Scalable Knowledge*, pages 195–201, March 2001.

[22] Markus Stumptner. An overview of knowledge-based configuration. *AI Communications*, 10(2) :111–125, June 1997.

[23] Markus Stumptner and Alois Haselböck. A generative constraint formalism for configuration problems. In P. Torasso, editor, *Advances in Artificial Intelligence : Proceedings of the Third Congress of the Italian Association for Artificial Intelligence AI*IA '93*, pages 302–313. Springer, Berlin, Heidelberg, 1993.